UMEÅ UNIVERSITY
Department of Computing Science
Catarina Andersson
c99can@cs.umu.se

5 June 2004

Software testing in the small enterprise Elpool i Umeå AB

Masters Thesis

Internal and external supervisors
Annabella Loconsole and Anders Olsson

ABSTRACT

The goals of this thesis are 1) to evaluate and choose testing tools and 2) to create a testing process.
The testing tools and process should be adopted to the small software company Elpool i Umeå AB. The work is motivated by the wish of Elpool to improve the quality of their software. This thesis will focus on product quality. Several methods can be applied to improve the product, for example fault prevention and fault detection techniques which are two parts of software testing.

The first goal has been fulfilled by evaluating several testing tools according to five criterias decided together with Elpool. Among others, they have to support the programming language Progress, which is the development language used by Elpool. The purpose of the tools is to act as help during the whole development cycle including the testing phases. The kinds of testing tools are test management, static and dynamic tools.
The second goal has been fulfilled by studing different test processes and adapting them to the needs and resources of Elpool. The test process is created by merging theories intended for big enterprises together with two test processes from [8] and [26].

The testing process and tools will be adopted by the company Elpool in the near future and any company with size and budget similar to Elpool is suggested to adopt the methods studied in this thesis.

PREFACE

This masters thesis is the result of twenty weeks full time work at the Department of Computing Science of Umeå University, Sweden. It is an indispensable part to receive a Master of Science in Computing Science and Engineering.

Contents

1 Introduction

1.1 Background

Software must be ensured to be of high quality since it performs more and more critical tasks which we are dependent on [17]. One way to increase the quality is product improvement. Product improvement can be done through software testing, which is focused on detecting faults and failures [7]. Establishing software testing in the lifecycle of a product is a required investment to survive in the market [19].

Software testing is the activity of finding faults. According to Myers [19],
"the purpose of testing is to discover errors. Testing is the process of trying to discover conceivable fault or weakness in a work product".
One big problem is "When do you know that everything is tested and is it really possible to test it all?".
It is proven that it would take hundreds of years (even if the rate of the testing would be hundreds per second) to test a program, which only performs the simple action of adding two integers (32-bits, leading to 2^64 test cases) [10]. What if exhaustive testing would be performed on a normal sized program? That is not possible.
Opposite software systems most systems fails in a reasonable small set of ways [7]. Therefore, detecting faults in software demands better detection techniques then other systems do.
To test a system as much as possible in a reasonable way, special techniques are needed. Some of them are presented in this thesis.
The goal of all research around testing today is to provide practical testing methods, tools and processes to help the engineers produce quality software at low costs. Software quality is becoming the most important part in the software industry. However, widespread testing techniques are not common in the industry today [17].
The thesis concerns how to behave if the enterprise is small and wants to test their software. Most of todays testing theories are developed for big companies with more personnel and resources in mind. This makes it very hard for small companies to apply the theories. The theories must be adopted to fit their purses [33]. On the other hand, it may be easier to establish new development and testing processes in a small company where the bureaucracy is less [23]. A mature testing process can then, without huge demands of investments, survive with a future expansion of the company [20].

1.2 Problem specification

At the market, a fast delivery of correct, new and changed software is a must to be able to compete with other similar companies. Therefore, Elpool i Umeå AB wants to adopt a testing process into their development cycle. It would improve the functionality of the software, reduce defect density and functionality against the rest of the already established complexity of programs. Elpool also wants to

adopt some testing tools to have support in their testing activities. Therefore, the goals of this thesis are
1) to improve sofware testing activities and
2) to introduce software software tools in the company
In order to reach the two goals, several questions need to be discussed.

Software testing activities:
When the developer has implemented the program, the developer wants a process to test all functions and relations to other parts of the system. Therefore a test routine, which will consist of a working plan and documentation of programs and relations must be developed. The plan will show how the programmer should do to improve the quality of the software.

Software testing tools:
The following questions have been formulated by Elpools employees. By introducing testing tools in the company, it is possible to get an answer to the quastions posed. The kind of testing tools, which are interesting, are ones that can completely or partly automate the testing activities.
How to let the developers work concurrently with the code? How to assure that the developers works with the latest version of the system?
How to make sure that the programmer have constant control over what is happening?
How to give the programmer signals of how the changes affects the rest of the system?
Which database tables will be updated in the changed program? Which relations do these have to other tables in the system? Which programs make these updates of the tables.
The programs, which make the changes, are they used by other programs? Which programs do the changed programs use?

1.3 Methods and results

The main results of this thesis are:

- An evaluation on two test management tools, static testing and dynamic testing tools according to five criterias. This result has been obtained by download and installation of evalaution versions of the tools in question. The proposed testing tools are selected according to how well they fulfil the five criterias.
- A test process containing the order to perform different activities. With Elpools size and prerequisites as a basis, the result is obtained by merging theories intended for large companies together with two testing processes.

The results can be generalized to other similar organisations having the same concerns.

In [19] can be read that testing personnel of today do their job and report the results. Still the management ships the product away. This happens when testing has a little authority in the organization. Another expectation for this report is that it can help software testing activities to get higher priority, showing that it is an essential part in the process cycle.

1.4 Outline of the thesis

The thesis is structured as follows: in chapter two an introduction to software testing theories and techniques is presented. Some of these theories will be applied to Elpool. Chapter three contains a definition of small enterprise. Also general problems of software quality in small companies are presented. Furthermore information about the company like background of the company, products and what they want to improve. In chapter four an evaluation with five criterias is made on test management, static and dynamic tools. The criterias are taken from [25] and chosen after discussions with the employees. In chapter five a testing process tailored for Elpool is constructed. The process suggests the order to perform some of the tasks presented in chapter two. The purpose of the testing process is to guide the developer to improve the quality of the software. The section also contains justifications why special techniques and methods are chosen to be applied at Elpool. Finally, in chapter six subjective concerns about the thesis, possible future work and the achieved result are discussed.

2 Software testing

Quality is one of the keystones in making the customer, the one who pays the bills, satisfied. If no quality assurance activities are included into the process cycle, the company is in great danger. Poor software quality depends on many things like low maturity of the used process and/or low product quality. Low product quality is mainly caused by faults. A fault is a mistake made by humans. The outcome from a fault, which is a mistake caused by a human, may result in a failure. The many different reasons why failures appears are [19]:

- The requirement specification may be wrong.
- The requirement specification contains something that is impossible to implement.
- The system design may contain fault(s).
- The program design may be wrong, it may use wrong algorithm(s).
- The program code may be wrong.

Software testing is focused on defect detection and it is one way to achieve quality assurance. Testing must be integrated into the total developing process, not just be a task in the end of the project [28].

Myers (1979) has a definition of software testing [19]. "The purpose of testing is to discover errors. Testing is the process of trying to discover conceivable fault or weakness in a work product". Other definitions of software testing are found in the literature. [17] says that testing is the process of finding differences between the expected output and the actual outputs. Another meaning of testing could be found in [36]. It tells us that testing activities tries to damage the software in a planned way.
According to [36], the following items are goals of software testing:

- Aimed at finding defects.
- Demonstrates the lack of quality.
- Shows the difference between the requirements and the developed system (verification).
- Checks whether the "right" system has been developed (validation).
- Establishes confidence in the product.
- Offers advise regarding the quality and risks.

Some suggestions about software testing are given by [19]:

1. The success of the testing is dependent of the quality of the test process used.

2. Around 50 percent of the defects are usually possible to trace to the requirement stage. If defects are detected in the same phase they are introduced, the cost will be reduced.
3. Many software testing tools are available, so the search after them can be worthwhile.
4. Someone must have the responsibility for improvements by planning and managing the progress.
5. Trained and skilled personnel are required for high quality software testing.
6. The testing attitude must be the will to find defects and to try to break the system.

Summarizing, software testing has some limitations. More then fifty percent of the total developing costs are possible to trace to the testing activities. Testing can not be able to show the absence of faults, nor the quality of the software product. So is it worth the effort of adding software testing activities into the organisation when it demands such a huge effort and does not even assures the lack of faults? The answer is yes. Software testing is a necessary activity to survive in the market. The pros survive the cons [17].

An interesting question is how to behave if you have millions lines of code to test. It is easy to understand that the need of designing and validating tests must be as controlled as the writing of the code.
In [19], the author states that complete testing consists both of verification and validation. White and black box testing are general validation techniques. White box testing examines the internal structure and is used to increase logic coverage of the program. In black box testing, the tester does not see the source code. Its purpose is to find differences between the expected and the actual output after entering inputs into the program. Validation is often called computer based testing. Another type of testing is verification. It involves inspections of the product and is therefore called human-based testing [19].

2.1 Test processes

To achieve high quality testing, a mature test process is a must. A test process describes all the activities that have to be performed from the very start to the end of a testing cycle.

Testing Maturity Model (TMM) states that a mature test process should include [8]:
Defined test policies: The test policies should be well defined, documented and supported by management and integrated into the organization culture.
Test planning: The test plan should include test goals, resources, design, test cases, schedules, costs, tasks and risks.
Test life cycle: A test life cycle contains a set of activities that are integrated into the software life cycle.
Test group: The test group should be independent. To motivate the testing

personnel, education opportunities should be given.
Test process improvement group: This group contains personnel from general process improvement group, software quality assurance group or testing group. It should apply incremental improvements and evaluate the techniques.
Test related metrics: Test data is collected and analysed. This is used at test process improvement.
Tools and equipment: The test improvement evaluates and proposes tools which can help the testing personnel with testing activities and collection of test related data.
Controlling and tracking: The test process is controlled by the test manager(s). The test manager(s) monitors and take action upon progress and occurred problems.
Product quality control: The testing should meet the quality standards. Stop testing criterias must be declared and defects must be tracked.

Another description of what a good test process must include can be found in [26].

- Test planning and management
- Dedicated testing environment and testing tools.
- Test case definition.
- Test automation.
- The handover to test department.
- Test execution.
- Test result analysis.
- Test reporting.
- Measurements of test effectiveness.
- Test process improvement.

Both ([8], [26]) have similar opinions of what a mature test process must contain.

Quality must be built into the software from the beginning of the development process. This is possible with an modified V-model (see figure 1). On the left hand side of the V-model, the requirements, design, coding phase and the testing activities related to them can be viewed. The right hand side shows test activities that must be performed after the implementation. In paragraph 2.4 a more detailed description of each activity can be found.

Agile testing has its own developed strategy and ideas. One of the goals of agile testing is simplicity, the most simple design and constructions will be used. Software or methods are applied first when they are needed. Communication is also one of the priority of agile testing. Person-to-person communication is used instead of written documents. Feedback is also one of the goals because it ensures that everything is on track. Finally, aggressiveness is the approach

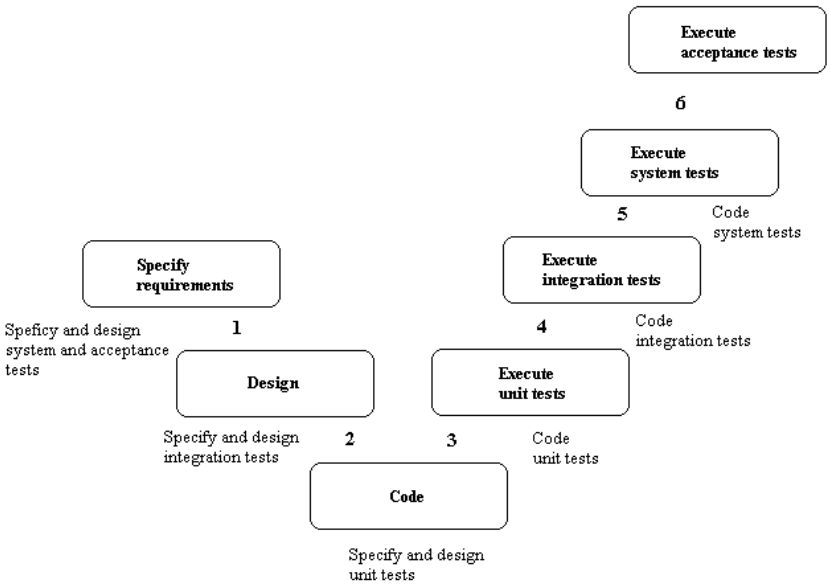

Figure 1: Modified V-model integrated with testing activities [12].

of wanting to move forward, which comes automatic when the three approaches explained above is working. By using these techniques, high quality software with high confidence can be developed in a quick way [11].
In agile methods small releases of the program are delivered during the development. The required changes are not seen as crisis but as part of the job [21]. The working methods results in the whole team taking responsibilities for the quality assurance including testing.
Another recommendation from the agile testing paradigm is addressed to the developers. They should construct small, testable parts so the testing part can start as early as possible [11].

2.2 Test planning and documentation

Testing is a sub process of a process and should be handled in the same way. To be as effective and provide as good results as possible, the need of a precise planning preparation is required.

The IEEE 829 standard for test documentation [26] suggests the following activities:

- Identify the scope to be tested
- Estimate the time, resources, people, hardware, software and tools.
- Provide the resources needed
- Provide test environment.

- Assign people to tasks.
- Define the schedule.
- Identify risks.
- Track progress and take corrective action.
- Provide regular test status of passed, blocked and failed tests.
- Re-plan if scope of the project changes.
- Insert evaluations if there are any lessons to learn from.

Example of a test schedule is shown in table 2.2.

Table 2.2: Test schedule [26].

Activity	Resource	Start date	End date	Comments
System test	Alice	2/4-02	6/9-02	Complete
Performance test	Bob	6/5-03	6/7-03	In progress

An independent group should act as test developers to avoid conflicts between personal responsible for the faults and the one who are supposed to discover them. It is not good to be the author and the tester of the code in the name of objectivity. To have an independent group of testers is also effective because the testing development can proceed concurrently with development of the product [28].
The testing should be stopped first when it is time to deliver or all tests pass. It is also dependent on the budget, how much money the organization has invested into testing [19].

Release plan and test case specification are the important parts of the project test plan, the other chapters will never be read by anyone [11]. The use of evaluation on the other hand is important at the end of the project to make place for possible future improvements. [21] further speaks about agile methods and their lack of detailed documentation of the project. The detailed documentation is replaced with close face-to-face communication with the customer and co-workers. The meetings about the parts of the product are short and informal discussions. The test status is presented in a public simple chart. The chart should show which parts of the product are tested and classified as finished.
According to [7], testing activities have to be documented in four different documents. They are master test plan, test case specification, test incident report and the test summary report.

The master test plan contains risk management. The aim of this plan is to get an overview of the whole testing phase. Interesting questions contained in the master test plan are "What kind of testing?", "How much verification and validation?" and "Should we consider acceptance testing?" [19]. An example of a test plan can be seen in figure 2.

1. Introduction. This section describes the objectives of the tests.

2. Relationship to other documents. Here will be explained how the tests are related to the functional and nonfunctional requirements.

3. System overview. System overview provides a explaination of the different components used in unit testing.

4. Features to be tested and not tested.

5. Pass and fail criteria.

6. Approach. The approach declare which techniques will be used.

7. Suspension and resumption. Explains what do do when testing is resumed.

8. Testing materials.

9. Test cases. Each test case is described in a separate report, the test case specification. The outcomes from the tests are documented in the test incident report.

10. Testing schedule. This part covers responisbility, staffing, training needs, risks and the schedule.

Figure 2: Test plan [7].

Also each activity in the V-model must construct a plan [19]. The verification plan considers verification activity, methods, quantity, risks associated with the areas that will not be inspected, tools, responsibilities and a schedule. Like for verification activities, validation activities also have their planning considerations. Example of items that such a document can have are detailed planning, testware design, , test execution, test evaluation, testware maintenance, test methods, facilities, test automation, testing tools, support software and risks. After completion of an activity, a document that reports the results of the performed activity must be constructed. A verification activity (left side) report can contain test item and version, number of participants, size of the materials inspected, preparation time, disposition of the software element, estimate of the rework and the rework effort completion date, defect list and defect summary.

As soon as the test plan is completed, it is possible to write the test case specification. The test cases usually contain a description with information about purpose of the test, setup required to execute the test case, inputs to the test case, the test procedure, expected outputs or results and a test summary whether the test passed, failed or blocked. Detailed information have to be given if the test did not succeed [19]. Also information which requirements, functions, methods and conditions that are tested can be included into the test case specification. A traceability matrix between test cases and requirements can look like table 2.2.

Table 2.2: Traceability matrix [28].

test	req no 1 (generate database)	req no 2 (change database)
1. add new record	X	
2. change field		X

When the test is performed, a test analysis report is constructed. It describes the result of the test. If failures occurred, discuss why it did and the following information should be given in a problem report form containing the subjects where, when, what was observed, consequences, how did it occur, why, how much did the user affect and how much did it cost. According to [18] no result reporting from the unit testing is needed. From functional testing, the interesting items are:

- How many functional tests are developed?
- How many tests are running correctly?
- Are the answers from the tests validated from the customers?
- How does the trend look like?

This information could easily be displayed in a graph, like the one shown in figure 3.

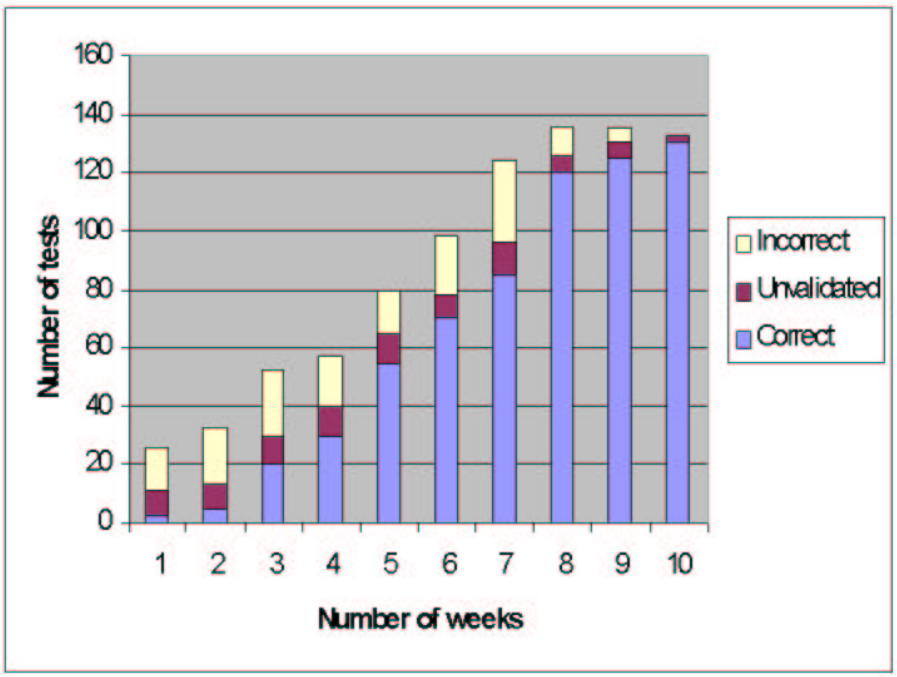

Figure 3: Graph showing results from functional testing [18].

2.3 Software verification testing

Verification finds faults at a static level without execution of the system. Verification testing is proven to be one of the safest and most cost-effective ways to quality improvement in both short and long term and is considered to be more effective then validation tests [19].

The central events in verification are reviews at which defects in the product are discussed and discovered. There are two types of reviews; walkthroughs and inspections. These reviewes consist of inspection of documents (requirements specification, source code etcetra) to find faults. A walkthrough is an informal meeting while an inspection is a more formal meeting [28].
Inspections have been shown to be extraordinary successful at detecting faults. An inspection consists of five steps. First a overview meeting is held where the author of the component presents the component to be inspected and its scope. A inspection meeting is held after the reviewers have had time to be familiar with the component. At the meeting, the inspection team comes with observations and comments. A moderator keeps the meeting in a good order. The author reworks the component and a follow-up is made to determine if a new meeting should be held or not [7]. Software verification makes it easier for the developers and testers later in the process because it deals with construction of testable requirements already at the beginning of the project [11]. The disadvantage of verification is that it assumes that the requirements are correct, which is not always the case [7].
Checklists for each action on the left side of the V-model (figure 1) can be found in Appendix A.

2.4 Testing small components

Before testing the complete system, each component must be tested. First to test are small components. The purpose of unit and integration testing is to ensure that the code implement the design properly [28].

2.4.1 Unit testing

Unit tests are done to verify that the component works properly (internal data structures, logic and conditions) in a controlled environment [7]. The three motivations of breaking the code down into components and make unit tests on them are making the complexity of the overall testing activity smaller, making it easier to correct faults and allowing parallelism when each component can be tested independent of the others [36].

The hard work lies in constructing unit tests [18]. Future testing of new versions of the software, to assure the new version do not brake anything old, will be simplified by rerunning the old test cases. New tests only need to be constructed for newly added parts.
When performing unit testing, three main steps are followed:

1. Examine the code.
2. Prove the code correct.
3. Test program components.

Examine the code
The first step has been described in paragraph paragraph 2.3 on page 11. What follows are the second and third steps.

Proving the code correct
The proof of correctness is to discover algorithmic faults. It is a way of formal understanding of the program and therefore forces all data structures and rules to be declared more precise. Symbolic execution is a variant of proving. It works as each line of the code is executed, the technique checks whether the state is changed or not.
However, proving demands much work, it takes more time to prove the code then to write it, large programs cause large logic diagrams, proving can not show that the implementation is correct and not to forget, all proofs are not correctly constructed. There are also many states and paths and the flow through loops is hard to follow.
Automated proving provided by a tool is hard to develop when it must be language dependent. It is very difficult to find a solution [28].

Testing program components
The arguments for testing program components instead of proving the correctness are that a test gives information about how program works in a dynamic mode and also demonstrates its functionality to the customer [19].

A test case is defined to be the input data used in testing a program [28]. A test case is successful when it discovers faults. The will to try to brake the product must therefore be the primary goal during the development of test cases [19]. Starting to develop test cases should be done as early as possible. It can be started as soon as the software life cycle starts. The only disadvantage of early development of test cases is that stubs and drivers may be changed as the requirements are changed [7]. In [36], advices and facts about test cases are given. It says, unplanned, non-reusable, throw-away test cases should be avoided unless the program to be tested is a throw-away program itself. To assure the correctness, the actual results must be compared to a predefined key or the expected output. The test cases must be written for invalid, unexpected, valid and expected, input conditions. Also test cases, which generate desired outputs, should be constructed. The test cases must be documented so they can be repeated later on [19].
In [19] suggestions about steps to be performed when it is time to execute test cases be read.

1. Select a test case.
2. Pre-run the setup and execution and make a analysis what the outcomes might be.

3. Record the results.
4. Determine whether failures depend on errors in product or in the testware.
5. Measure the internal logic coverage.

According to [13], at construction of *automatic tests* a good advice is to use the same programming language used for writing the source code, because the test team are familiar to that language. Not all tests should be dependent of the GUI because GUI-tests are rather fragile when it comes to changes of the application. Further could be read that automatic tests with advantage can be constructed before implementation of the source code. During the development, the developer can use the tests to assure that the application works. Since both constructing automatic tests and perform manual testing, have their different disadvantages, the suggestion is to develop small testing programs.

Testing the program components can be done with help from validation methods [19]. As mentioned earlier, validation methods consists of black and white box methods. According to [24], a combination of black and white box testing is recommended. Using only one of them results in incomplete testing.

The *black box technique* supplies the program with inputs and observes the output. The source code behind the user interface is not bothered. The big disadvantage is to not know which parts of the source code is covered by the tests. Black box testing can result in untested areas of the source code [28].
Several black box techniques are available, equivalence partitioning, boundary values and error guessing etcetera.
Equivalence partitioning is a systematic process, which identifies a set of interesting classes of input conditions to be tested. To reduce the number of test cases, each class covers a large set of other possible tests. The identification of the equivalence classes will be done as follows [28]:

1. Identifying equivalence classes (EC) for each external input:
   (a) If the input is specified by valid values, define one valid EC and two invalid. Example if one to five are valid values, take two, minus two and seven.
   (b) If the input specifies the number of valid values, define one valid EC and two invalid Example if the input requires two to six fields, write four, one and seven.
   (c) If the input defines a set of valid values, define one valid EC and one invalid. Example if the input requires the names Andy, Charles and Lisa, try Lisa and George.
   (d) If the input specifies "must be", define one valid EC and one invalid. Example if the first character must be numeric, define one that starts with a number and one that does not.
   (e) Subdivide the class if there is a reason to believe that the elements in the EC will not be handled in an identical manner.
2. Identifying test cases.

(a) Assign a unique number to each class.

(b) Until all valid classes have been covered by test cases, write a new test case covering as many of the uncovered classes as possible.

(c) Until all invalid classes have been covered, write a test case that covers one and only one of the uncovered invalid classes.

(d) If multiple invalid classes are tested in the same test case, some of those tests may never be executed because the first test may mask other tests or terminate execution of the test case.

Boundary-value analysis is a refinement of equivalence partitioning.

- Rather then selecting any element in an EC, edge elements are selected. Example if 0.0 to 1.0 is valid, choose 0.0, 1.0, -0.9 and 1.1
- Output conditions are also explored by defining output ECs.

The disadvantage of this technique is that the edge elements might be hard to identify.
Error guessing is an ad hoc approach based on experience to identify test cases, which are likely to find errors. An idea is to make a list of possible errors and then construct test cases based on them. Defect histories are useful when applying this method. According to [19], likely items to test are empty or null lists/strings, zero instances, blanks or null characters in strings and negative numbers.
Test cases can in advantage randomly be generated over night to save time. Randomly selecting input cases between minimum and maximum is a reliable method [24].
Suggestions of test cases to performing black box testing can be found in Appendix A.

The *white box technique* examines the internal logic of the program. It is easy to end up testing what the program does instead of what it should do which is the major disadvantage of white box techniques [28]. Examples of white box techniques are instruction, branch and condition coverage. At least one should be used to achieve the best result of finding faults [6].
Path testing is a white box testing technique. The technique exercises all paths though the code at least once. To use path testing, first a flow graph will be constructed with nodes as executable blocks and associations as the flow of control.
Path testing is developed for imperative languages. Object oriented languages often has shorter methods which makes flow control less important. Polymorphism in object orientation also makes path testing a inefficient technique where more tests are needed to cover up the many possibilities of bindings [7].
Suggestions concerning white box testing are described in [37]. See Appendix A.

After the tests are executed, a test evaluation is made which contains test coverage based on function coverage matrix and requirements coverage matrix. It

shows if further tests are required. Based on the number of detected errors, their nature and area, a product evaluation is made. Based on these two evaluations, a test effectiveness evaluation is computed. The tester must be able to answer questions about quality of the software, stability, if the product is ready to be released, effectivness of the performed testing, number of known problems and how much testing remains to be done [19].

2.4.2 Integration testing

As shown in figure 1, once each component is tested individually, time is to combine them and test them together. Stubs and drivers used earlier in the unit testing stage could be faulty so there is no guarantee that the separate components actually work. A test driver is a partial implementation, which depends on the tested component. It calls the component under testing. A test stub is a partial implementation, which the tested component depends on. It is the part that the component under testing will call. By using a carefully thought strategy, producing unnecessary stubs and drivers will be avoided [7]. Examples of integration testing methods are bottom up, top down, big bang and sandwich integration.

Bottom up integration
Each component at the lowest level of the design is tested first. The ones to be tested next are the ones that call the previously tested ones. This process is repeated until all components are included. The bottom up approach is a good way of isolating the problem. On the other hand, the top-level components, which are the most important ones, are the last to be tested. The top level often controls the system and it might be difficult to test the lower parts of the system excluding it.

Top down integration
The method top down integration lets the controlling unit (highest level), be tested first. After it is completed, the components called by the components already tested, are included and tested. This procedure is repeated until all components are included.

test A -> test A, B, C -> test A, B, C, D

One problem with this strategy is the writing of the stubs. They must allow all possible conditions to be tested. Also the required number of stubs may be high.

Big-bang integration
Big-bang integration uses the technique to first test all components individually and then mix them all together as a final system. This approach is not recommended for any kind of system. It requires a lot of stubs and the causes of failures are hard to find [28].

Sandwich integration
Sandwich testing combines the top down and bottom up strategies. By grouping

the subsystem into three layers, target layer, a layer above the target and a layer below the target, top down and bottom up testing can be done in parallel. Testing the top layer with the target layer and bottom up testing on the lower layer and target layer makes top down integration. No stubs or drivers are needed to be written for the top and lower level since they use the actual target layer.
This strategy allows early testing of the interface components. The disadvantage on the other hand is the lack of individual testing of the target layer before it is integrated with the two other layers.
Because of this problem, a modified sandwich testing strategy is developed. The difference is to test the three layers separately before integrate them with each other. This requires construction of stubs and drivers for the target layer [7].

2.5 Testing the whole system

The purpose of system testing is to ensure that the system does what the customer wants it to do. This is done executing the black box testing methods [28]. System testing is the testing activity after unit and integration testing where functional and non-functional requirements are tested. In [19], system testing is explained as being the most difficult testing part. This is the process of showing that according to the requirements, the system does not do what it is intended to do. Not to forget, a successful test finds the differences between the requirements and the actual behaviour of the system. The steps in figure 4 should not be tested by the person or organization that developed it [19].

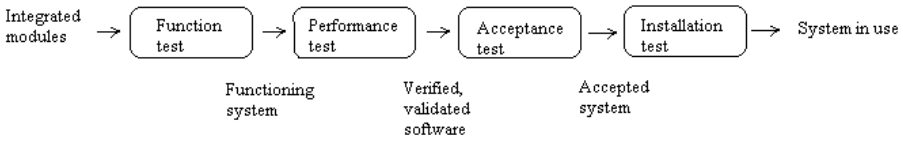

Figure 4: The steps in the system testing process [28].

After the functional and performance tests, the system is called to be verified as it operates the way the designers intends it to do. Next step is to compare the system with the expectations of the customer by reviewing the requirements. This leads to a validated system.

Sometimes large systems are impossible to test as whole. Instead the system could be phased into subsystems. This approach also makes it easier to locate faults and narrow the search after them. This solution requires a development plan, which defines subsystems, drivers, stubs and from who, how, where and when the tests will be performed.

2.5.1 Function testing

Function testing focuses on functionality and ignores the system structure. Function tests checks if the integrated system performs its functions as specified in the requirements.
It is possible to begin function testing before the entire system is complete. A cause and effect graph could be constructed. It reduces the number of test cases but is not practical for systems that include time delays, iterations or loops nor where the system reacts to feedback from some of its processes to perform other processes [28].

The agile testing area has similar ideas and recommendations. The purpose of functional tests is to assure the management and the customer that the system does what it intends to do [18]. Confidence is easily made to the tests by letting the customers provide test data. Every time an upgrade of the system is made, new tests should be written. In addition, the old tests need to pass. The tests must be developed in parallel with the application implementation so the developers achieve fast feedback. This leads to the advantage that faults can be corrected as fast as possible.

2.5.2 Performance testing

Performance testing is testing made on non-functional requirements [7]. Performance testing includes security, accuracy, speed and reliability. Hardware is an essential part of the performance, therefore hardware engineers may be part of the performance testing team. Different kinds of performance tests are:

- Stress tests: stress the system to its limit over a period of time (maximum number of users etcetera).
- Timing tests: tests the times to respond to the user and to perform an action. Timing tests are usually done together with stress tests.
- Volume tests: handling of large amounts of data. Concerns to be checked here could be the size of the data structures and files.
- Configuration tests: evaluate all possible configurations to make sure that each satisfies the requirements.
- Compability tests: assure that the system cooperates with other kinds of systems like for instance databases or servers.
- Regression tests: assures that the new system works at least as good as the old one.
- Security tests: checks the availability, integrity and confidentiality.
- Environmental tests: how does the system react towards heat, humidity, motion, chemical, moisture, portability, electrical or magnetical fields and disruption of power?

- Quality tests: reliability, maintainability, availability, mean time to failure, mean time to repair, average time to find and fix a fault.
- Recovery tests: response to faults of data, of power, of devices and see if it recovers correctly.
- Maintenance tests: investigates the need of diagnostic tools and procedures to help finding faults.
- Documentation tests: are the requirements documentation, user guides and technical documentations written?
- Human factors tests (usability tests): checks the display screens, messages and report formats.

2.5.3 Acceptance testing

Acceptance testing compares the end product to the current needs of its end users. The customer performs this task by running the system under a period of time [19]. This is the reason why acceptance test also is called customer testing. In this phase, the customer wants to determine if the developed system really meets its expectations. Different types of tests could be [28]:

- Benchmark test: the customer prepares a set of cases, which represent typical situations.
- Pilot test: the system will be tested in a experimental way. No test cases are prepared but the system is set in its everyday work, which is a less formal method.
- Alpha test: let the system be tested by users from the development company.
- Beta test: after a while when the system has been executing, let the customer perform new tests on the system.
- Parallel testing: the new system operates in parallel with the old one. The user can easily see the advantages and compare the two.

Usability testing lets the users work with the product and observe how it behaves. This should be done as early as possible in the development cycle and two to three times during the development [19]. Pilot testing is common in functionality tests. It is performed in the target environment executed by the users. Examples of pilot testing are alpha and beta testing. The difference between alpha and beta tests and usability tests is the behaviour of the user will be recorded when using usability tests [7].

2.5.4 Installation testing

A test performed after the installation is completed is called installation test. Installation tests allow the customer to run the complete system in the right

location to assure that the system is working. This kind of testing is only needed if no acceptance tests were done in the right target location [28].

2.6 Regression testing

Regression tests are tests applied to a new version or release of the system to verify that it still performs the same functions like the older version did [28]. Regression testing can only be performed if old test cases are saved [16].
Regression testing is one of the most expensive parts during life cycle of the software. Estimations made, show that one third of the total budget goes to regression testing. To decrease the costs, selective methods are available which choose from a set of test cases, the ones that should be rerun. Prioritised test cases are those that maximize the code coverage, minimize the cost and minimize the run time [17]. Regression testing includes the following steps:

1. Insert new code.
2. Test functions known to be affected by the new code.
3. Test essential functions of the old version.
4. Function test the new version, which involves test cases from the former version.

According to [4] even small changes to the software may damage a lot. This is the reason why regression testing is such an important task.
The need of automated regression testing is high because of the impact it does on the possibility of delivering the updated, tested product at short notice. Software updates can be made quickly but delivering retested and high reliability products demands help from automatisation. This is a matter of competition and the time and money invested in training and developing of test scripts will pay off in the future.
To achieve total retesting is not always realistic. Instead only the parts of the product, which are critical, should be retested. Of course, if time is available, the rest of the system should also be retested. The steps to explore are identifying the lines of code which will be executed and then eventually develop new tests which cover the newly added source code.
To assure the new code does not make impacts on the old performance, comparison checks have to be done. Examples of comparisons are snapshots of the old and the new screen appearance and the look of databases.

With the help from a set of old test cases, comparisons between old and new behaviours can be made. Three outcomes are possible. First, no changes are discovered, second, some differences are discovered but they are not important and third, major changes are found. In the two last outcomes, the reason could be a correct result of a bug-fix. In that case, the expected output must be changed. Once all test cases are passed, meaning no differences from the old behaviour, the new version is accepted [38].

3 Software development in small enterprises

A small enterprise does not have more then 50 employees, has an annual balance sheet not exceeding five million euro and 25 percent and not more of the voting rights are allowed to be owned by a single or multiple companies which are not classified as small enterprises [29].
According to [32], small enterprises are the factory of new ideas and serve as a ground for the economy. They are also the ones that are most sensible to quick changes in the industrial economics.

3.1 Software process improvements in small and large enterprises

[23] has developed a software process for small projects including few team members. What defines a small process is that it may not have professional process writers, trained inspection moderators and other supporting development members. Defined roles and assigned responsibilities are more and more important the bigger the project size. The more exactly an organization wants to measure quality attributes, the more specific process activities are required. The communication between the personnel is more informal and the bureaucracy is not present. The development phases are similar to the V-model (see figure 1). The following necessary roles can be held by one or many persons and one person can hold many roles. The roles are, according to [23], necessary for small software enterprises.

- **Conceptualizer** who has has the original idea for the project.
- **Customer** who finances the project.
- **User expert** who will use the system after delivery.
- **Manager** who coordinates the project.
- **Architect** who owns and creates the architecture.
- **Developer** who creates the source code.
- **Tester** who performs the testing activities.

It is easier for big enterprises to apply methods of software process improvements. According to [33] such improvements constructed for large companies, must be adapted to small enterprises to fit their purses. Another general problem in small enterprises is the smaller fund [24]. CMM could be taken as an example. CMM requires too many roles and personnel possible for a small company. Further, the roles includes too much responsibility. A CMM adapted to small enterprises can be found in [20].
Adding testing routines into the current software development cycle is an improvement of the whole software process. Testing Maturity Model (TMM) is a guide to organizations that want to assess and improve their testing process. It

is a complement to CMM addressed to test managers, test specialists and software quality personnel. This model focuses on the testing process contratary other existing process evaluations and assessments like CMM, ISO 9001, Bootstrap and SPICE [8].
Not to forget is the advantage of small size. Like said before, communication between the personnel is more informal and the bureaucracy is not present [23].

3.2 Software testing in small and large enterprises

Not all theories presented in chapter 2 can be applied in a small enterprise. It is likely that some adaptions must be made.
Theories applied for software testing in big companies states that testing should be performed by a independent test group [19]. This is not applicable in small companies where the lack of personnel and available funds are present. Instead, testers are the ones who also develop the code [23].
To involve automatic testing tools provided by testing vendors in a testing process is easier for big companies. The reason is again the available budget [24].
Establish development and testing processes, which fits the resources, in a small enterprise is easier then in a big one. The improvements can then, without huge demands of investments, survive with a future expansion of the company [20].

3.3 Improving testing methods and tools at Elpool

Elpool is a small software developing company, established 1988. Today Elpool has seven employees where two deals with the administrative tasks and the rest serve as developers. Their work consists of adapting the existing systems to each customer.
The name of the main software system is Guru and it performs tasks like work planning, the handling of materials, calculating, the record of time, flexitime and personnel administration for companies in the electricity distribution business.
The software and databases are developed using development kit provided by Progress Software, version 9.1 Progress Provision Plus. Progress is a procedural programming language but can also generate source code by drawing graphics. The definition of a program is a progress file (.p or .w) containing source code. The number of existing programs is around two thousands where the size of one program can vary in between a couple up to ten thousands of lines.
At the moment, before this thesis is applied, the programs are tested by comparing the actual and expected output with each other.
The people at the company would like to adopt a process that ensures the quality of the product and increase the control of the product to the programmer.

4 Evaluation on testing tools applicable for Elpool

This chapter first introduces general information about software testing tools. Then two test management tools are compared according to a set of metrics. The metrics are also used to perform a evaluation of static and dynamic testing tools. Finally, a list of summary containing the most suitable tools for the company Elpool.

4.1 Software testing tools

Testing tools are important because they can make the test process easier [26]. Automated tests can replay actions and reproduce faults, which is a big advantage. Using testing tools makes the execution of the testing part much faster and can be done every day. If the product is large and hard to debug, there is a bigger value in investing automated testing. Automatic tests are also much better then humans at detecting precise bugs such as decimals [22].
Automatisation helps but can not replace the human function. Tools are not free from bugs. One big disadvantage introducing new tools is that it seldom comes without complications. Difficulties installing them at the first place or using them are examples of problems which ends up in a waste of time and money [28].
The dream of automated testing would be just to enter the whole program into some device and then let it examine it after failures. Research teams of today are working to make this dream come true. An user profile is needed (telling how the program should work), a random input generator and a good test oracle. Current working tools that could be named as automated are capture/playback tools and tools requiring script containing input and expected outputs and then produces reports considering the failures [16].
Unfortunately all tests are not always worth automate. Issues to have in mind before implementing automated tests are:

- How much will the automated tests cost if developing them manually?
- How long life time can the automated tests have?
- Is it likely automated tests will find additional bugs?

There exist a lot of different types of testing tools [28]. Examples of areas where they can assist the testing team are code analysis, test execution, test management and test case generation.

Code analysis tools
Static and dynamic testing tools are two kinds of code analysis tools. Static analysis is performed when the program is not executing. It has different kinds of strategies, which are:

- Code analyser: looks for proper syntax like recording where a variable was declared the first time.

- Structure checker: notes the logical flow, for instance mark statements, which are never used and note presence of branch in a loop. Structure checkers helps the developer to arrange the code, to get information about how the different programs are dependent of each other.
- Data analyser: reviews the data structures, declarations, component interfaces and that arguments are passed properly.
- Sequence checker: example of an operation could be to check that all files are opened before they are modified.
- Cross reference: provides information to the programmer like if a variable is changed, it will show where it must be changed in other places.

Tools that help to make reviews and inspections are *complexity tool*. They can help identify complex, high-risk areas. *Code comprehension tools* help to understand unfamiliar code by tracing dependencies, logic, provide graphical representations and identify dead code. *Syntax and semantic tools* find errors that a compiler would miss [12].

Dynamic analysis is performed when the program is executing. This method is useful when several parallel operations are being performed concurrently like in real-time systems.

Test execution tools
Test execution tools runs test cases automatically. Examples of test execution tools are capture and replay tools which simulates keystrokes and compare expected and actual output. They are useful to verify, that after a change in the source code, everything is still correct.

Test management tools
Controlling versions and releases of a system is one way to achieve software quality. Software testing and software quality have close relationship to each other [7]. Concluding, version controlling tools can be defined as software testing tools and act as a help during the management of the testing activities.
Deltas, separate files and conditional compilation are three ways to control version and release control. A disadvantage with separate files is if similar changes must be done to both versions, leads to double work. A solution is to make a main version and let all other files work as variants of this one. The only file which need to be stored is the difference file, also called the delta. With this approach, changes are only made to the main version. A delta also has its disadvantage. If the main version is lost, all versions are lost. The handling of deltas is also hard. If the main version adds lines in the code, the line numbers do not correspond to the delta file anymore.
Conditional compilation is when a single code component addresses all versions. By using the compiler, it is possible to determine which statements that applies to which version. Unfortunately the code will be very complex and hard to read.
It is complicated when more than one developer makes changes to the same component. The figure 5 shows one of many problems if two failures occur during testing. Concurrent version system tools (cvs) take care of this concern.

1. **Two failures have occured during testing.**
2. **X and Y are assigned to fix them.**
3. **X and Y discovers that the root of the problem is the same.**
4. **X removes the wrong function, makes the changes and replaces the new version in the library.**
5. **Y, working on the original version, replaces X version with Ys after the update is done. This leads to Xs change will remain undone.**

Figure 5: A possible occurred problem at correction of faults [28].

Test case generators Test case generators are tools, which act as help at construction of test cases. They generates test cases so all possible situations will be covered by test cases.
Structural test cases generates test cases to use in white box testing like path, branch or statement testing.
Other test case generators base their construction on data flow, states of variables or randomly create sets of test data [28].
According to Progress, no test case generators applicable for Progress 4GL exist. Therefore, the only solution for Progress users is to develop tests on their own. Testing performed with expensive automatic testing tools, do not end in finding more faults [24]. Therefore, the decision not to include and evaluate test case generator in the thesis is taken.

4.2 Criterias used in the evaluation

The kinds of testing tools that will be evaluated are test management (cvs) and code analysers (static analysis and dynamic analysis) tools. Several metrics to evaluate testing tools are found in [25]. They may be a useful guidance at selecting suitable testing tools. The metrics are:

Human interface design: Human interface design considers if the tester must switch alot between mouse, keyboard and type alot of data into input fieldes.
Tool management: Tool management is about possibility to concurrent work like read results from a test, add a test or modify tests.
Ease of use: Ease of use is defined as the mean time learning the tool and operational time of frequent and casual users.
User control: User control includes the possibility of the interaction with the tool.
Test case generation: Test case generation is measured by the ability from the tool of producing test cases without any input from the tester.
Support from the supplier: Support from the supplier includes help desks via telephone, email, on-line users group and user manuals.
Return of investment: Return of investment includes aspects as saved time

using the testing tool and increased quality of the product.
Features support: Features support is the features like integration between the database and development tools and summaries of findings.
Reliability of testing vendor: Reliability of testing vendor includes factors like if the vendortool has been applied in real world applications and number of customers.

Elpool was asked to consider about the importance of each metric. The metrics "Human interface design", "Tool management" and "Reliability of testing vendor" will not be included into the evaluation. Even if they are satisfied, they are not important to the company.
Every tool must supply Progress Development Environment and work on Windows 2000, which is the operating system on the development machines.
When required, the tools and programs were downloaded, tested and evaluated with the criterias above as basis.

4.3 Test management tools

The following is an evaluation of two tools, Roundtable and Wincvs. The concerned objectives are how to handle the problems of concurrent manipulation of the same product and how to deal with different versions of the system.

4.3.1 Roundtable

Roundtable allows version control. Is specially integrated with the development language Progress. Among others it can save the state of an database. It also has the basic features of check in/out and commit. Check out is the first command to type to get an own working copy of Roundtable environment. Check in means adding a new file into the cvs environment. Commit adds the specific file into the Roundtable environment and is labelled with the version number. Upon a commit of a file, it is possible to see what was changed, by who, the reason of the change and information like which procedure reference an include file or database field. Except from dealing with version handling, Roundtable tracks development tasks, dependencies, deployments, encryption and compilation [3].

4.3.2 Wincvs

Wincvs is a free software solving version controlling tasks. Wincvs is applicable for all kinds of files. It also has the basic features of check in/out and commit and update [5]. Check out is the first command to type. It creates a own working copy of the common repository. Check in means adding a new file into the repository. Commit adds the specific file into the wincvs environment and is labelled with the version number. Update merges the own copy of the file together with the file in the repository [5].

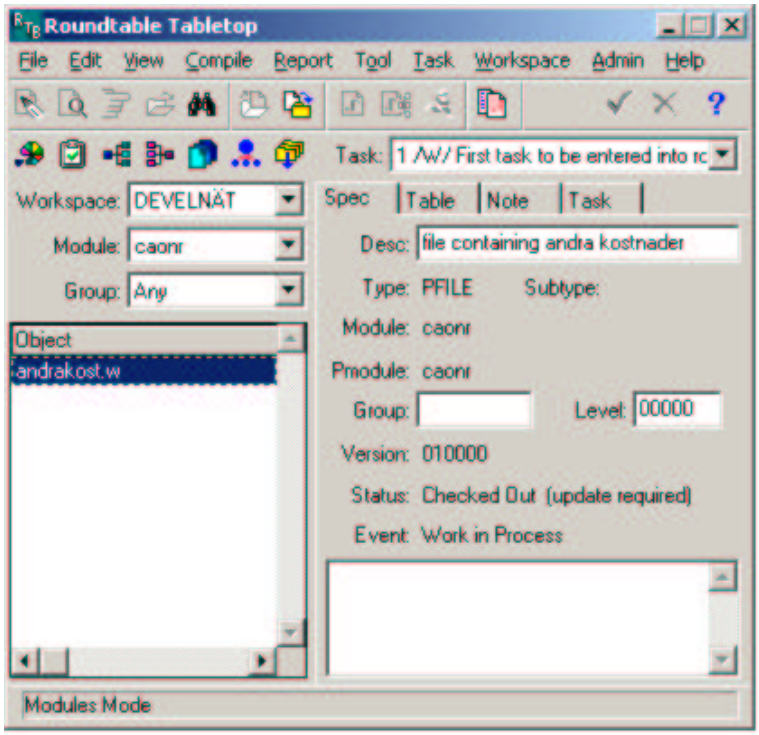

Figure 6: Start window of Roundtable.

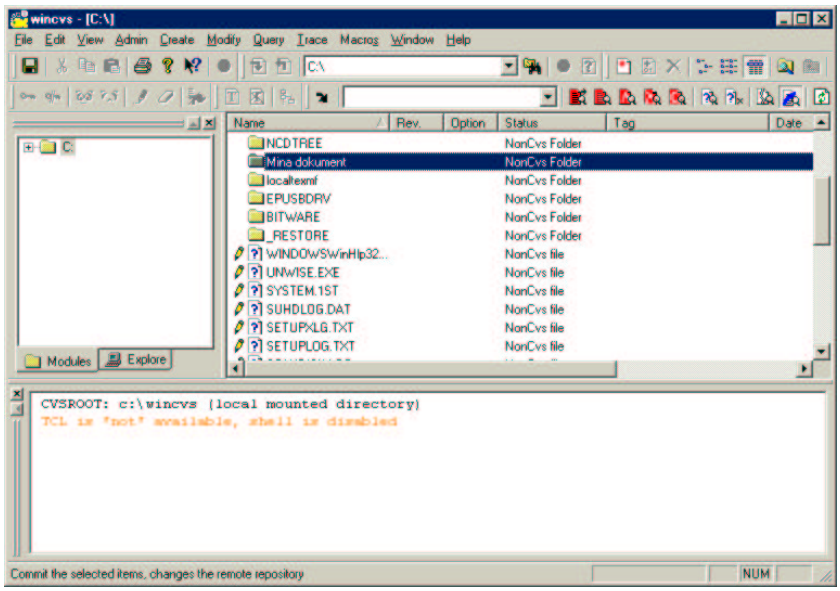

Figure 7: Start window of Wincvs.

4.3.3 Evaluation of the version controlling tools

In this section the criterias listed in paragraph 4.2 on page 25 will be applied to the two test management tools. The result of the evaluation can be seen in table 4.3.3. A "Yes" means that the tool satisfies the criteria. At places were "do not know" is given, it is hard to determine a correct objective answer.

Table 4.3.3: Evaluation of cvs tools.

Criteria	Roundtable	Wincvs
Ease of use	Do not know	Do not know
User control	Yes	Yes
Support from the supplier	Yes	Yes
Return of investment	Do not know	Yes(free software)
Features support	Yes	No

Summarizing, the tools seem similar. The main difference between the two tools is that Roundtable has the possibility of saving the look of the database.
The ease of use is very subjective but it is likely that support is needed in the installation and start up phase. The criteria "Test case generation" has not been evaluated in this case because it is not applicable

4.4 Static testing tools

The tools presented in this section can be defined as static testing tools. The tools find out how the dependencies look like and what kind of connections the different parts of the program complex have to each other. The initial question, achieving a structure and understanding of the Guru source code, will be concerned. To get an apprehension if the tools are good or not, a survey based on the metrics in paragraph 4.2 on page 25 is made.

4.4.1 Compile statement

To achieve the detailed structure, cross referencing tools are needed. According to the Progress company, this problem can be solved by using compile statements found in the Progress library. Compile statements allow parameters at compile time that generates lists concerning which tables and fields in different programs that are dependent of each other.
XREF is a compile statement, which has a number of advantages that helps documenting and understanding of the source code by finding dependencies between functions, modules and applications. The syntax of the XREF compile statement is:

COMPILE progress_file XREF outfile

where progress_file is the file to compile and outfile the file where the results can be viewed [30]. An example output file is shown in Appendix B. One of

the goals of this thesis was to know which tables a program updates, look for UPDATE in the output file.

LISTING is another compile statement that shows how the nesting of the include-files looks like [15].
The generated output contains information like the name of the file containing the compiled procedure, date and time, the number of each line and block number. The syntax of LISTING is:

COMPILE progress_file LISTING outfile

For an example of the output file, see Appendix B. Explanations of the generated data can be found in [30].

4.4.2 Start up parameter

With the feature start up parameter, it is possible to track which procedures are called or which they call when the system gets interaction from the user.
Using -yx in the startup file, the information is generated in an output file called proc.mon, placed in the working directory. An example and further information could be found in [31]. The real output file generated after execution on Guru, could be found in the Appendix B.

4.4.3 Table relations report

The objective how the dependencies between the database tables look like can be solved with table relations report (TRR). It is a built in tool which interacts with the database. The feature can be used by invoking the menu from the data dictionary

Database->Reports->Table Relations->All Tables.

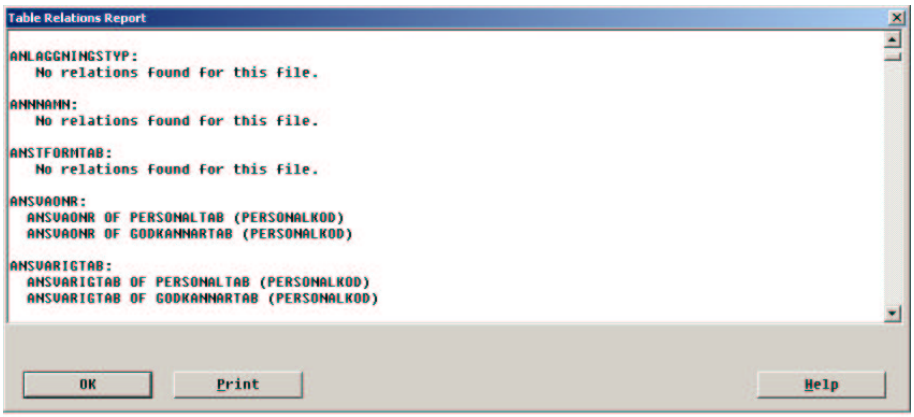

Figure 8: Report showing the dependencies between the database tables.

4.4.4 Progress File Manager

Progress File Manager (PFM) is a structure checker tool, which creates a tree containing the files of a system. The tree is built according to how the files (or programs) are dependent of each other, see figure 9. Progress File Manager finds these dependencies by analysing the code of the include files [27]. Progress File Manager can for instance be used in integration testing when three layers of the system must be identified. It is also possible to see which other programs the changed program is dependent on or uses. The contratary relation, find which programs that uses the changed program, the approach of parsing the file tree could be applied.

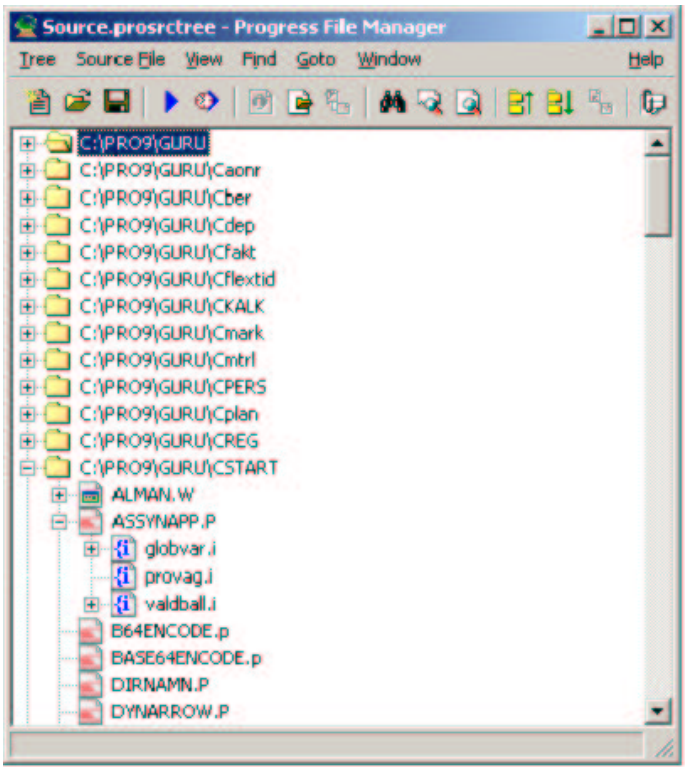

Figure 9: An example of how the program Progress File Manager views the dependencies.

4.4.5 Evaluation on the static testing tools

In this section the criterias declared in paragraph 4.2 on page 25 will be applied to the static testing tools. The result of the survey can be seen in table 4.4.5. A "Yes" means that the tool satisfies the criteria.

Table 4.4.5: Evaluation on static testing tools.

Criteria	PFM	XREF	LISTING	-yx	TRR
Ease of use	Yes	Yes	Yes	Yes	Yes
User control	Yes	Yes	Yes	Yes	Yes
Support from the supplier	No	Yes	Yes	Yes	Yes
Return of investment	Yes	Yes	Yes	Yes	Yes
Features support	No	Yes	Yes	Yes	Yes

Summarizing, XREF, LISTING, -yx and TRR accomplish all criterias except for generating test cases, neither should this be required by static testing tools, which are not test case generators.
All tools are included into the development environment except for Progress File Manager.
Progress File Manager only fulfils 50 percent of the criterias. Still, the tool can be recommended because most of the important criterias are accomplished. The criteria Ease of use in mind is accomplished. This leads to the conclusion that the criteria Support from the supplier may not be important.

4.5 A dynamic testing tool

tt-viewer is a tool that serves to detect defects. The criterias listed in paragraph 4.2 on page 25 will be applied. The result of the evaluation can be seen in table 4.5.

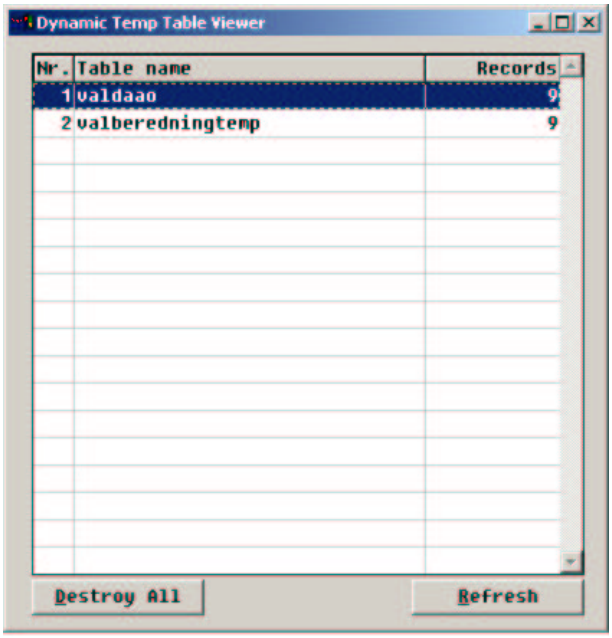

Figure 10: Two discovered temporary tables using tt-viewer.w.

tt-viewer.w helps when programming temporary tables [35]. A temporary table is a temporary Progress object. Example of a temporary object can be a copy of a sharp table from the data base. A temporary table that is shared between

different programs contains the same value. A handle is a pointer to one or many temporary tables. To avoid memory leakage, the temporary table must be removed when it is no longer in use. The deletion of a temporary table is done by removal of the handle to the temporary table. This may not always be the case, it might be forgotten during the development phase. tt-viewer finds all active temporary tables in the specific procedure tt-viewer is executed from. If the number of existing temporary tables after entering the program, leaving it and then entering it again increases, it is an indication of incorrect removal of the temporary table and actions must be taken. Figure 10 views the number of temporary tables used during executing of the procedure.

Table 4.5: How well tt-viewer.w fulfils the metrics.

Criteria	Fulfilled
Ease of use	Yes
User control	Yes
Test case generation	No
Support from the supplier	No (may be possible)
Return of investment	Yes (free software)
Features support	Yes

A "Yes" means that the tool satisfies the criteria.
Summarizing, tt-viewer.w fails on the requirements of generating test cases and support from supplier. Support from supplier may not on the other hand be important because of the uncomplicated use.

4.6 Suggested testing tools for Elpool

The suggested testing tools provide different tasks. The table below gives a quick comprehension about their nature.

Table 4.6: Performance of the testing tools.

Tool	Traceability	Structure of the code	Memory leakage	Version control
tt-viewer.w			X	
Start up parameter	X			
Compile statement	X			
Progress File Manager		X		
Roundtable	X			X
Wincvs				X
Table relation report	X	X		

After the evaluation of the tools, Elpool is suggested to use the following tools.

Test management tool
Roundtable and Wincvs perform the same tasks. The only differences are that Roundtable can perform traceability actions like compile statements and save

the state of the database. The traceability action is not required by a version controlling tool when that can be covered by other tools presented in the evaluation. Another difference is that Wincvs is a free software, which is preferable. Wincvs advantages beats Roundtable and therefore Wincvs should be chosen.

Static testing tool
The static testing tools LISTING and XREF fulfil the same criterias. XREF is to be preferred to LISTING because LISTING only presents complex information instead of, showing what happens with the fields of the database.
-yx should be used because it shows the dependencies between the programs of the system at execution time.
Table relations report is the only tool which shows the dependecies on the database level and it is easy to use.
Progress File Manager is also to prefer when it shows the dependencies of the programs at a static level.

Dynamic testing tool
tt-viewer must be used because it is the only found memory leakage testing tool applicable to the Progress language.

5 A test process for Elpool

This chapter is a proposal to a testing process tailored for the company Elpool. The steps of the tailored testing process applicable to Elpool are taken from established test processes [8] and [26] presented in paragraph 2.1 on page 5. The adaptations are developed from discussions with the company in question and recommendations from the sections in chapter 2.
The objective of this chapter is a process consisting of stepd which help the programmer to improve the quality of the software.
As in the previous chapters, it is important to remember that fault free software is never possible to ensure. What will come is a proposal how to increase the quality of the software.

5.1 Necessary adaptations to the resources of Elpool

Elpool is a relatively small size company. Therefore, all methods intended for big enterprises presented earlier in the thesis (see chapter 2) are not applicable to Elpool. Table 5.1 shows which activities are included into the test process.

It is very difficult for a small enterprise to have an independent testing team because of the lack of resources [23]. Still the professional feeling *a successful test finds faults* must be present [19].
Elpool wants to document the testing activities. It must be available for everybody at the company, and be simple so it is easy to perform and understand. In [21] is said that close face to face communication with the customer is replaced with documentation. Close face to face methods also exists at Elpool and therefore some documentation can be skipped. Only release plan, test case specification and the results from functional testing are important for the company. This documentation approach except for only reporting the results of functional tests may suit Elpools.
No result reporting from the unit testing is needed according to [18]. Only results of functional testing is interesting.
Verification activities are necessary and cost effective and must be included into the test process [19].
Black and white box methods contain good advises when it comes to testing small components and should therefore be used.
Exercising every path in the code is very hard from the economics perspective. Still the approach of testing as many branches as possible should be used.
Elpool should not prove the code correct. The advantages of proving are hardly not existing [28].
The modified sandwich integration technique is the most developed integration technique with least disadvantages [7].
Among the system testing methods, function, performance and installation testing are possible for Elpool to apply. Acceptance testing on the other hand is not possible because the customer is not integrated into the test process at Elpool.
According to [13], test cases should be constructed before the implementation is done. Much of the code to be tested is already implemented and therefore this approach is impossible to realize.

Every now and then, Elpool creates new versions of their products and wants to assure that the changes do not affect the old system. Therefore, Elpool should most of the time perform regression testing. However regression testing is not possible at this point because no tests for regression testing exist. Therefore construction of test cases for future regression testing must be done.

Table 5.1: Testing activities applicable at Elpool.

Activity	Section	Applicable
Apply a validated test process	2.1 page 5	Yes
Independent testing team	3.2 page 22	No
Construct a test schedule	2.2 page 8	Yes
Release plan	2.2 page 8	Yes
Test case documentation	2.2 page 8	Yes
Test incident report	2.2 page 8	Yes
Test status	2.2 page 8	Yes
Master test plan	2.2 page 8	No
Activity planning	2.2 page 8	No
Activity summary report	2.2 page 9	No
Traceability matrix	2.2 page 9	No
Test analysis report for unit tests	2.2 page 10	No
Test analysis report for functional tests	2.2 page 10	Yes
Verification testing	2.3 page 11	Yes
Testing subsystems with black box methods	2.4.1 page 13	Yes
Testing subsystems with white box methods	2.4.1 page 14	Yes
Exercise every path of the code	2.4.1 page 14	No
Proving the code correct	2.4.1 page 11	No
Modified sandwich integration	2.4.2 page 15	Yes
Function testing	2.5.1 page 17	Yes
Performance testing	4.6 page 32	Yes
Acceptance testing	2.5.3 page 18	No
Installation testing	2.5.4 page 18	Yes
Construction of test cases before implementation	2.4.1 page 12	Yes
Perform test evaluation	2.4.1 page 12	Yes
Regression testing	2.6 page 19	No
Test case generation tools	4.1 page 23	No
Static testing tools	4.1 page 23	Yes
Test execution tools	4.1 page 23	No
Test management tools	4.1 page 23	Yes

5.2 The resulting test process

The following test process is a unification of two test processes [26] and [8] (see paragraph 2.1 on page 5). So, why should this particular test process be good? A mature test process is to prefer. A mature test process is defined to manage planning, staffing, directing, controlling and organizing components. Further, a mature test process must be supported by management and act as a part of the whole software lifecycle. It must also be easy to understand and possible

to improve [8]. This test process fulfils the described criterias and is defined as complete. The steps Handover to test department, Test automatisation and Measurements of test effectiveness are removed. Handover to test department does not fit into Elpools organisation because they do not have a separate department for testing. The step Test automatisation is handled in step one and Measurements of test effectiveness is not possible because no coverage tools working on Progress code were available at the time of constructing the thesis.

Step 1: Plan the test
Test planning is the first step in a testing process. Paragraph 2.2 on page 7 contains the template IEEE 829 which explains the parts belonging to test planning. Under IEEE 829, there is a suggestion of what a test schedule should contain. Also a risk management is required. One item in IEEE 829 is to explore required testing tools. Install and get familiar to the suggested testing tools.
One suggestion how to perform regular test reporting is to establish a public board where the status of the testing cases can be viewed. This board will work as a release plan of the system/component. The table containing which parts of the program that is tested and completed, name of the tester and start and planned end date, should be updated every day when the work is finished [21]. In table 5.2 is an example schedule during the Unit testing activity.

Table 5.2: Example of release plan.

Tester	Program	Start date	End date	Status
Alice	program1.w	2/2-04	10/2-04	In progress
Bob	program2.p	4/2-04	9/2-04	Completed
Trudy	program3.p	11/2-04	17/2-04	Not started

Step 2: Determine test environment
To be able to perform the tests, the testing team should find out the required testing environment. A testing environment is the required hard and software to perform the testing activities. For instance which work stations which testing tools are needed. This step is similar to the previous one.

Step 3: Produce test case content
The test case content is a document which contains the following steps:

1. Purpose of the test.
2. Setup required to execute the test case.
3. Inputs to the test case.
4. The test procedure, how to execute the test.
5. Expected outputs or results.
6. Summary (passed, failed or blocked):

Test case documentation is relevant because the future implementation of the test cases in step 4. Advises and recommendations what the test cases should test can be found in Appendix A.

Step 4: Execute tests
The testing activities described below are taken from figure 1 on page 7. The stop testing criterias are dependent on the budget, how much money the organization has invested into testing. Other stop testing criterias are determined by the time to deliver the system or by the completion of the tests [19].

Perform verification testing (see paragraph 2.3 on page 11).
The checklists described in Appendix A are all applicable for Elpool except for the code verification check list. In an interview performed at Elpool, the following items are chosen to perform code verification:

1. Data reference errors
   (a) When indexing into a array or string, are the limits exceeded?
   (b) Are storages given the right attributes?
   (c) Are data structures used in multiple routines defined the same way everywhere?
   (d) Is storage allocated for referenced pointers?
   (e) Are the units allocating smaller then the units of storage addressing?
2. Other checks
   (a) Does the program check the input validity?
   (b) Are there memory leaks (for instance are all handles removed).
   (c) When ending a procedure in a .p program located on the AppServer, are there any locked tables?

Informal discussions between the assigned testers and developers can result in short summaries [21]. If there are newly detected or remaining defects, the status of the public testing board, table 5.2, must be updated. This will be done after each verification activity independent if it is informal or not.

Perform unit tests (see section 2.4.1 on page 11).
One unit is defined to be a file with extension .p or .w. Test cases for unit testing can, if the code to test is not yet written, be written before the implementation. *Assign personnel to write the test cases.* Parallel work is easily done. One tester can start to test file one and another file two and so on. Implement the test cases produced in step 3. Use Progress as a language when all developers know that. *Save the tests for future regression testing. Make the tests as small check programs [13].* After their construction phase, the tests should be executed and the results be documented to be able to be compared to the expected output.
See section 2.4.1 on page 11 for more details.
The following steps are decided to be included into the testing process.

1. Test input and output parameters from procedures.
2. Check that the functions AVAILABLE and ASSIGN are used together.
3. All checks must be done against sharp tables from the database.

4. Check the number of temporary tables does not increase during execution. This is done with the program ttviewer.w.
5. After a program on the client calls a program located on Appserver "PERSISTENT", check that the lock is released when the transaction ends. This is accomplished by, after completed updates, check that "RELEASE" is executed when any transaction ends.

Perform integration tests (see 2.4.2 on page 15).
Perform modified sandwich integration, which is the best developed method handling integration tests.
A suggestion is to run Progress File Manager to know how the files are dependent of each other and identify three layers.

Perform system tests (see paragraph 2.5 on page 16).
System testing means functional, performance, acceptance and installation tests. The required documentation is the results of functional testing, see figure 3.

After an update of the system, according to the customers changed requirements and needs, perform the following activities:

1. Make a backup of the program of the customer.
2. Change the system.
3. Perform regression testing to assure the system still works.
4. Use a cvs to commit the new system. It is now possible for all developers to make an update and achieve the latest version to work with.

Step 5: Document the results of the tests
Document if the test passed or failed. If it failed, document it in a test case incident report that contains information about when, what was observed, consequences, how, why and if the user (developer) affected the failures.
Review the results and intensify testing at areas where errors occurred frequently.
An analysis of the results is required to get a measurement of the effectiveness and to assure that progress is made. An advise how to present the progress of test case results can be viewed in figure 11.

Step 6: Evaluate the quality
When all testing activities are done, the tester must be able to answer the questions about the current quality of the product, the stability of the product, if it is ready to be released, the product quality compared to earlier versions, if the performed testing was effective, if there are any known problems and if there is remained testing to be done. These kinds of questions are necessary to be able to improve the test process.

Step 7: Deliver
At a database update, it is important to deliver the software at right time, neither to late nor to soon. This consideration must be taken if the data base

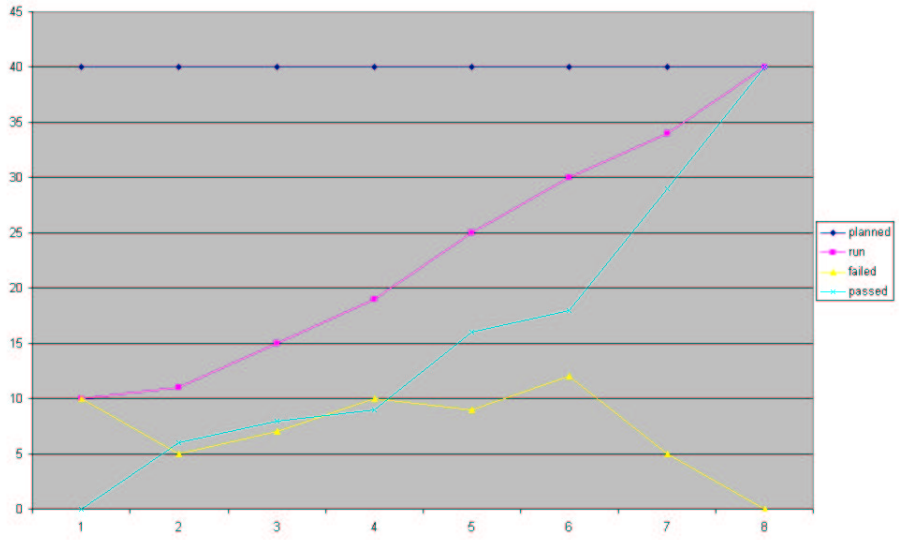

Figure 11: Test case results [26].

structure is changed when there is a risk that old programs may use an old version of the database, which no longer exists.
These steps should be repeated every time a new product is developed. If an update is made, only perform regression testing.

5.3 Summary of the test process

Table 5.3: Summary of the test process.

Steps	Description	Input	Output
Step 1	Plan the test.		test plan, test schedule, knowledge in the testing tools , public reporting board
Step 2	Determine testing environment.	software to be tested	list of needed software
Step 3	Test case documentation.		explained test cases
Step 4	Execute tests.	software to be tested	tested software, short summaries
Step 5	Document of the test results.	recorded results	test case result report, test case incident report graph showing the progress
Step 6	Evaluate the quality.	software to evaluate	list of obtained criterias of the software
Step 7	Deliver.	updated system	updated system

6 Conclusions

In this thesis a study of software testing techniques is done. Particular attention are paid to testing process and testing tools. In chapter 4 several testing tools are evaluated and then the ones which are most suitable for Elpool are chosen. In chapter 5 a testing process is created for the company based on processes suggested by [8] and [26] (see chapter 2.1 for more details).
The following table shows which parts of the initial objectives that have been solved and in which section the solution can be found.

Table 6: Fulfilled objectives.

Objective	Fulfilled	Paragraph	How
- A test process suggesting the developer how to improve the product quality.	Yes	5 page 35	Test process
- Documentation of programs and relations.	Yes	4.4 page 28	Progress file manager
- Testing tools which completely or partly can automate the testing activities.	Yes	4.4 page 28 and 4.5 page 31	static and dynamic testing tools
- An assurance that everybody works with the newest version.	Yes	4.3 page 26	Wincvs
- Give the programmer constant control over what is happening.	No		
- Give the developer signals of how the changes affect the rest of the system.	Yes	4.4 page 28	static testing tools
- Know which database tables will be updated in the changed program.	Yes	4.4 page 28	XREF
- Relations between the database tables in the system.	Yes	4.4.3 page 29	Table relations report
- Know which programs makes the updates of the tables.	No		
- Know if the program that make the changes uses other programs.	Yes	4.4 page 28	static testing tools
- Know how the changed program can be reached from other programs.	Yes	4.4 page 28	Progress file manager

Constant control is not achieved. All proposed tools have to be run by a human and can not work as alarms when something is changed in the system.
It is possible to trace the changes made by a program into the database but the opposite way, trace which program that changed the value of an tables field, can not be done.

Automatisation of test cases creation, requires a tool, which is not found. The obvious limitation of this thesis is the selection of testing tools. For sure, other tools are available but for time reasons, they have not been evaluated. Techniques concerning construction of automated test cases for Progress are not present when this masters thesis is written, either could Progress recommend any special testing tool vendor. Hopefully such techniques are discovered in the future to ease testing for companies. Other structure checkers were not found.

Unfortunately structure checkers are extremely language dependent. Progress is commonly used but not widely spread as many other development languages. This decreases the possibility of finding appropriate tools and in particular structure checkers.
Another reason not finding a appropriate testing vendor, was the difficulty getting response from the vendors. The only one contact established was Compuware. The reason for the lack of interest is impossible to answer. Could it be the small size of the enterprise equal less funds equal less probability of being a reliable customer? I hope not. The solution of this problem is to constantly search for new testing tools. The communication forum Progresstalk is a good source at finding such new inventions applicable for Progress.

The lack of developed published testing processes and advices applied for small enterprises is big. Therefore this proposed test process with its tools and adaptions may also be useful to other similar, small enterprises, using Progress as a development environment or not.

It is likely that Elpool will change over to dynamic programming in the future where OO is exploited more. The thesis does not include these concerns because they are not considered to be important. Future work will be to explore more about dynamic testing when the company will pass on to other developing strategies like smart objects.

When the proposed test process will work well, it could be increased with further methods and documentation. This approach is required in a future expansion of the company. A suggestion would be try to reach higher levels of TMM. Instead of focus on detection of faults, the strategy of prevent them could be applied.
No test cases are developed so far. Therefore, the need of selecting test cases from a set of test cases at regression testing not yet is current. This technique will be required in the future.
A stimulating task for the future will be to write a program solving the problem with the relationships between the programs and tables in the database. It would combine cross referencing and the structure checker tool. For instance, if a table in the database is changed, there would be no need for execution of start up parameter or compile statements, producing huge output files, which are hard to parse. Instead the name of the table is typed into the program and the program solves the dependencies and relations it has to other tables and programs. Only the necessary information would be displayed.

To validate the results of this thesis, new counts of MTBF should be done and compared with the old numbers, available before the results of this work are applied in the organisation. An easy action would be to count the numbers of requirement changes from the customers before introducing the tools and procedures suggested here. After that, perform a new count. At the time, Elpool gets around six to seven changes of requirements per week. The obvious expectation is to decrease that number.

My supervisor at Elpool has validated the results. According to him, the proposals are possible to realize in the company.

7 Acknowledgements

I am grateful to Elpool i Umeå AB. Without you, this thesis would not be possible. I specially want to thank my external supervisors Anders for our conversations about the company. Further I appreciate Germund who gave me advises about tt-viewer and Progress File Manager.
I am also grateful to Mats Nilsson at Progress for giving me advices regarding their product and available tools. It saved me a lot of work.
Annabella, my supervisors at the department of computing science, Umeå university, thank you for being such a support searching literature and finding out a structure in my masters thesis.
At last but not the least, Jenny, for being so kind helping me correct my English grammar.

8 Glossary

CMM: Capability Maturity Model. A model for improving software processes.
Component: a part of the system that can be isolated for testing. Objects, group of objects or a subsystem are examples of components [7].
Correction: a change to a component. Corrections can introduce new faults [7].
CVS: Concurrent Version System [1]
Error: a human mistake in performing a software activity [28].
Failure: a occured difference between the requirements and the actual behaviour of the component. The source to a failure is error(s) [28].
Fault: an incorrect step, process or data definition in a computer program. The outgrowth of a error. (May lead to failure.) [28]
Functional requirement: describes an interaction between the system and its environment, how the system should react to different stimulis [28].
Mistake: a human action that produces an incorrect result [19].
MTBF: Mean Time Between Failures [28].
Oracle problem: testers test their output against the expected one (foreseen by the "oracle") [9].
Process: a particular course of action intended to achieve a result [2].
Regressive testing: re-execution of some or all the tests [19].
Subtest: objective is to detect errors [19].
System: any organized assembly of resources and procedures united and regulated by interaction or interdependence to accomplish a set of specific functions [14].
Test case: a set of inputs and expected results trying to cause failure at the component [7].
Test driver: a partial implementation which depends on the tested component. It calls the component under testing [7].
Test point: interrogates the value of a variable, performs one or more specific actions used to invoke with the system to be tested [19].
Test stub: a partial implementation which the tested component depends on. It is the part that the component under testing will call [7].
TMM: Testing maturity model. A model to improve test processes. A complement to CMM [8].
Validation: process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. Validation involves executing the software and is also called computer-based testing [19]. Validation ensures that the developer builds the right product [28].
Verification: process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Verification is a process of evaluation, reviews and inspections. Verification is human testing which involves looking at documents of paper [19].

References

[1] Concurrent version system. Last visited May 2004. http://www.cvshome.org/project/www/docs/ddCVS.html.

[2] Process. Last visited May 2004. http://www.thefreedictionary.com/process.

[3] RoundTable Tugboat Software. Last visited May 2004. http://www.roundtable-tsms.com.

[4] Software Regression Testing. Last visited May 2004. http://www.visibleprogress.com/software_regression_testing.htm.

[5] Wincvs. Last visited May 2004. http://www.cvsgui.org.

[6] Bokelman, W. J. *Application Testing In The Real World.* Compuware Corporation, 1999.

[7] Bruegge. Testing. Tech. rep., May 1999.

[8] Burnstein I., Suwannasart T., C. Developing a testing maturity model Part 2. *Crosstalk* (1996).

[9] Chen, T. Semi-proving: an integrated method based on global symolic evaluation and metamorphic testing. *Software engineering notes*, 4 (2002), 191–195.

[10] Cigitallabs. Software testing. Last visited May 2004. http://www.cigitallabs.com/resources/definitions/software-testing.html.

[11] Crispin, L. XP Testing Without XP: Taking Advantage of Agile Testing Practices. Last visited May 2004. http://www.methodsandtools.com/archive/archive.php?id=2.

[12] Daich G., Gordon P., R. B. Software Test Technologies Report. *STSC* (1994).

[13] Finsterwalder, M. Automating Acceptance Tests for GUI Applications in an Extreme Programming Environment. *ACM Press*, 114–117.

[14] FS-1037C. System. Last visited May 2004. http://www.its.bldrdoc.gov/fs1037/dir-036/_5255.htm.

[15] Gaither, R. Progress Programming Tips. Last visited May 2004. http://www.westnet.com/ gsmith/ppt/ppt26.htm.

[16] Hamlet, D. *The Engineering of Software, Technical Foundations for the Individual.* Addison Wesley, 2001.

[17] Harrold, M. Testing: A roadmap. *ACM Press* (2000).

[18] Jeffries, R. Why aggresive software development calls for radical testing efforts. Last visited May 2004. http://www.xprogramming.com/publications/SP99

[19] KIT, E. *Software testing in the real world, improving the process*. Addison-Wesley, 1995.

[20] LARYD A., O. T. CMM för småföretag, Nivå två. *UMINF-00.10, Umeå University*, Aug. (2000).

[21] MARICK, B. Agile Methods and Agile Testing. Last visited May 2004. http://www.testing.com/agile/agile-testing-essay.html.

[22] MARICK, B. When should a test be automated? ACM Press, 1998. Paper presented at Quality Week.

[23] MELISSA L. RUSS, J. D. M. A software developing process for small projects. *IEEE Software*, 9 (2000), 96-101.

[24] MENZIES, T. When to test less. *IEEE Software*, 5 (2000), 107-112.

[25] MICHAEL J.B., BOSSUYT B.J., SNYDER B.B. Metrics for Measuring the Effectiveness of Software-Testing Tools. Tech. rep., Nov. 2002.

[26] OT'REGEN, G. *A practical approach to software quality*. Springer Verlag New York, Inc., 2002.

[27] PEACOCK, C. Progress File Manager. Last visited May 2004. http://freespace.virgin.net/chris.peacock/software/progressfile.htm .

[28] PFLEEGER, S. *Software Engineering Theory and Practice*. Prentice Hall, 2001.

[29] PIEMONTE. Definition of small enterprise (1996). Last visited May 2004. http://www.piemonte.org/inglese/eib/definition.php3.

[30] PROGRESS. *Progress Language Reference Syntax A to F, Version 8*. Progress Software Corporation, 1997.

[31] PROGRESS, LTD. *Progress Startup Command and Paramenter Reference*. U.S, May 2002.

[32] RANDEN, J. Europeisk stadga för småföretagen. *NUTEK* (2003).

[33] RICHARDSON, R. Software Process Improvements in a Very Small Company. Last visited May 2004. http://sqp.asq.org.

[34] SQATESTER.COM. Testing tips. Last visited May 2004. http://www.sqatester.com/testingtips/blackbox/index.htm.

[35] VAN DAM, P. Dynamic temp table viewer. Last visited May 2004. http://v9stuff.com/tt-viewer.w.

[36] VAN VEENENDAAL, E. Test Process Improvement. Presentation, 2001. Presentation at European Software Control and Metrics Conference, London, UK, April 2001.

[37] VASIGA, T. How to test programs. Last visited May 2004. http://www.math.uwaterloo.ca/

[38] WILKINSON R. Regression Testing. Tech. rep., July 1998.

9 Appendix A

The following are some suggestions for verification testing. Checklist for requirements

- Complete. All items needed to specify the solution to the problem have been included.
- Correct. Each item is free from error.
- Precise and clear. Each item is exact and understood.
- Consistent. No item conflicts with another item in the specification.
- Relevant
- Testable. During program development and acceptance testing, it must be possible to determine whether the item has been satisfied.
- Traceable. Each item can be traced to its origin environment.
- Feasible. Each item can be implemented with the available techniques, tools, personnel, cost and schedule.
- Manageable. The requirements are expressed in a way that each item can be changed without large impact on other items.

Checklist for functional design

- When a structure is described in words, try to sketch a picture of the structure being described.
- When a calculation is specified, work at least two examples and give them as example in the specification.
- Watch for vague words, such as some, sometimes, often, usually, ordinary, customarily, most or mostly.

Checklist for internal design

- Does the design document contain a description of the procedure?
- Is there a model of the user interface to the computing system?
- Is there a high-level functional model of the proposed computing system?
- Are the major implementation alternatives represented in the document?

Checklist for code

- Data reference errors.

- Data declaration errors. Are there variables with similar names?
- Computation errors. Is the target variable of an assignment smaller then the right hand expression?
- Comparison errors.
- Control flow errors. Is there a possibility of a too early loop exit?
- Interface errors.
- Input/output errors. Are there grammatical errors in the output text of the program?
- Portability. How is the data organized?
- Other checks. If compile warnings are produced, check each.

Complete verification check lists are shown in [19].

The following are some suggestions for black box testing.

Programs containing numbers and strings should test [37]:

- nice values of different types (positive, negative, zero)
- boundary conditions
- maximum, minimum
- outside maximum and minimum
- prime and even numbers

Programs containing strings should test:

- delimiter problems (missing or too many)
- mixed cases
- input which is to long for string
- input which has white space or other delimiter

In programs containing editable fields, try to type [34]:

- valid characters
- invalid characters
- valid minimum
- valid maximum
- valid middle

In programs containing required fields, try to type null strings or no data at all are filled to see how the system reacts.

In programs containing text or memo fields, try to:

- type in character limit
- cut
- copy
- paste

All spellings in warnings and error messages or dialogs should be checked. Also invoke all menu items and their options.
Check how the placement of objects appears on the screen. Then test to open all menus and check their items.

Try all functionalities for example:

- File+Open+Save
- If the system is right mouse clicking sensitive.
- If it is possible to resize, minimize, maximize and restore the windows.
- The scrollability from the keyboard and auto scrolling if it is present.
- How does the keyboard and mouse navigation work like highlighting and drag and drop?
- Check if it is possible to print in landscape an portrait modes.
- Check F1 and the help chapter. Does the tab key help to navigate in all dialog boxes and menus?

Basic compatibility such as 16 bit OS (Win 3.x, Win95, OS/2, WinNT3.x) and 32 bit OS (Win95, Win98, Win2000, WinNT4.x, UNIX).

The following are some advises for white box testing [37].

Programs containing files should test:

- file exists and contains correct data
- file exists but data is of wrong type or format
- file exists but is empty
- file exists but is corrupt
- file does not exist

Programs containing logics should test:

- boolean values of 0/false, 1/true and something like 2/hi
- nested statements roughly
- all conditions in all case statements

Programs containing loops should test:

- entering the loop if the condition is true
- if the exit values are expected
- if the loop is exited at the correct iteration
- whether the loop body executes zero, once or multiple times

Programs containing data structures and pointers should test:

- ordered data structure (first element added/removed, middle element added/removed, last element added/removed)
- unordered data structure (empty structure, single item, multiple items, full structure, duplicate items)
- pointers (nil, pointer is not nil, two pointers pointing to the same object, pointer to a list of multiple objects)

10 Appendix B

The command
COMPILE LISTING allarbt.p outfile
produces an output file outfile. Part of the outfile are:

```
...RU\WTID\allarbt.p                  12/04/2003 13:49:04   PROGRESS(R) Page 1

{} Line Blk
-- ---- ---
      1     /*ALLARBT.P*/
      2     {PHMT.I}
 1    1     /*PHMT.I*/
 1    2     {PERSONALTEMP.I}
 2    1     /*PERSONALTEMP.I*/
 2    2     DEFINE {&NEW} {&SHARED} TEMP-TABLE personaltemp  NO-UNDO
 2    3        FIELD AKTIV             AS LOGICAL
 2    4        FIELD ANSTALLNING       AS CHARACTER
 2    5        FIELD ANSTNR            AS CHARACTER
 2    6        FIELD ANSVARIGTIDR      AS CHARACTER
 2    7        FIELD BEFATTNING        AS CHARACTER
 2    8        FIELD BEREDSKAPSAVTAL   AS CHARACTER
 2    9        FIELD BOXEN             AS CHARACTER
 2   10        FIELD BRAVO             AS LOGICAL
 2   11        FIELD DELTID            AS LOGICAL
 2   12        FIELD EFTERNAMN         AS CHARACTER
.
.
.
...RU\WTID\allarbt.p                  12/04/2003 13:49:04   PROGRESS(R) Page 9

     File Name       Line Blk. Type Tran            Blk. Label
-------------------- ---- --------- ---- --------------------------------
...RU\WTID\allarbt.p  164 Procedure No   Procedure rullkoll_UI
...\WTID\RULLVECKO.I    7 Do        No
...\WTID\RULLVECKO.I    9 Do        No
...\WTID\RULLVECKO.I   13 Do        No
...\WTID\RULLVECKO.I   21 Do        No
...\WTID\RULLVECKO.I   23 Do        No
...RU\WTID\allarbt.p    0 Procedure No
    Buffers: RT9.RULLVECKO
             RT9.RULLSCHEMA
             RT9.RULLPERS
             RT9.ARBETSTIDTAB
             tidutbuff
             RT9.VECKOARBETID
             RT9.VECKOARBAV
             RT9.PERSONALTAB
```

```
             valperstemp
             tidut

...RU\WTID\allarbt.p   61 Do        No
...RU\WTID\allarbt.p   68 Do        No
...RU\WTID\allarbt.p   73 Do        No
...RU\WTID\allarbt.p   81 Do        No
...RU\WTID\allarbt.p  104 Do        No
```

The command
COMPILE XREF allarbt.p outfile
produces the output file outfile. Parts of the outfile are:

```
c:\pro9\GURU\WTID\allarbt.p c:\pro9\GURU\WTID\allarbt.p 1 COMPILE allarbt.p
c:\pro9\GURU\WTID\allarbt.p c:\pro9\GURU\WTID\allarbt.p 1 CPINTERNAL ISO8859-1
c:\pro9\GURU\WTID\allarbt.p c:\pro9\GURU\WTID\allarbt.p 1 CPSTREAM iso8859-1
c:\pro9\GURU\WTID\allarbt.p c:\pro9\GURU\WTID\allarbt.p 2 INCLUDE PHMT.I
c:\pro9\GURU\WTID\allarbt.p c:\pro9\GURU\WPERS\PHMT.I 2 INCLUDE PERSONALTEMP.I
c:\pro9\GURU\WPERS\PHMT.I c:\pro9\GURU\WPERS\PERSONALTEMP.I 2 STRING "personaltemp"
12 NONE UNTRANSLATABLE
c:\pro9\GURU\WPERS\PHMT.I c:\pro9\GURU\WPERS\PERSONALTEMP.I 2 STRING "AKTIV"
5 NONE UNTRANSLATABLE
c:\pro9\GURU\WPERS\PHMT.I c:\pro9\GURU\WPERS\PERSONALTEMP.I 2 STRING "ANSTALLNING"
11 NONE UNTRANSLATABLE
c:\pro9\GURU\WPERS\PHMT.I c:\pro9\GURU\WPERS\PERSONALTEMP.I 2 STRING "ANSTNR"
6 NONE UNTRANSLATABLE
.
.
.
c:\pro9\GURU\WTID\allarbt.p c:\pro9\GURU\WTID\allarbt.p 21 ACCESS SHARED regdagnamn
c:\pro9\GURU\WTID\allarbt.p c:\pro9\GURU\WTID\allarbt.p 21 ACCESS SHARED regvnr
c:\pro9\GURU\WTID\allarbt.p c:\pro9\GURU\WTID\allarbt.p 21 STRING "ONS" 3 NONE
TRANSLATABLE
.
.
.

c:\pro9\GURU\WTID\allarbt.p c:\pro9\GURU\WTID\allarbt.p 149 SEARCH RT9.ARBETSTIDTAB
ARBTIDKOD
.
.
.
c:\pro9\GURU\WTID\allarbt.p c:\pro9\GURU\WTID\allarbt.p 21 UPDATE SHARED regdagnamn
c:\pro9\GURU\WTID\allarbt.p c:\pro9\GURU\WTID\allarbt.p 21 UPDATE SHARED regvnr
```

The -yx start up parameter
produces a proc.mon file. A part of the proc.mon file is:

```
Thu Dec 11 09:20:53 2003

Procedure call statistics: 09:22:31
```

Caller	Callee	Load Size	Calls	Rd Bytes	Reread	Time
<top>	_edit	4505	1	4505	0	0
_edit	_adeload	11199	1	11199	0	20
_edit	_prostar	11945	1	11945	0	10
_prostar	_login	6769	1	6769	0	15332
_edit	_setcurs	746	1	746	0	0
_edit	_proedit	294076	1	294076	0	7972
_proedit	_toollic	8118	1	8118	0	420
_toollic	_locdwb	1450	1	1450	0	0
_toollic	_locard	1440	1	1440	0	0
_proedit	_tmpfile	1388	2	1388	0	10
_proedit	_kvlist	3100	1	3100	0	0
_proedit	_ossfnam	4003	4	4003	0	0
_proedit	_adehelp	5331	1	5331	0	10
_proedit	_mnkvals	7237	1	7237	0	90
_mnkvals	_setcurs	746	2	0	0	0
_proedit	_setcurs	746	7	0	0	10
_proedit	_adeevnt	3311	3	3311	0	0
_proedit	_osfext	942	2	942	0	0
_proedit	_wfrun	1741	1	1741	0	0
_proedit	_runcode	28003	1	28003	0	10
_runcode	p84902r	480	1	0	0	0
p84902r	gurust	1351	1	0	0	50
gurust	WSTARTST	484	1	0	0	10
WSTARTST	WSTART	100022	1	0	0	5127
WSTART	EGENBENA	10129	1	0	0	130
WSTART	STYRFORE	5486	1	0	0	50
WSTART	EGENSTART	9153	1	0	0	61
WSTART	TYPSKAPFAKT	7392	1	0	0	70